

R Programming Basics – Hands-On Practical

Igor Ruiz de los Mozos

NOTE to render instal `quarto install tinytex`

Programme: Master's in Biomedical Engineering • **Module:** Advanced Bioinformatics

Lesson R Fundamentals, Data Wrangling & Visualisation

Table of Contents

1. Overview & Learning Objectives
 2. The RStudio Interface
 3. R Programming Basics
 4. Exploring Objects & Structures
 5. Wrangling with **dplyr**
 6. Reshaping with **tidyr**
 7. Visualising with **ggplot2**
 8. Full Workflow on `biomed_data.csv`
-

1 Overview & Learning Objectives

Introduction

This practical introduces the foundational concepts of R programming and its application in bioinformatics. R is a powerful language for statistical computing and data visualization, widely used in biomedical research.

Goals


By the end of this session, you will be able to:

- Navigate the RStudio interface and understand its key components.
- Write and execute basic R code for data manipulation and visualization.
- Use **dplyr** verbs to filter, summarize, and join datasets.
- Reshape data using **tidyr** for better analysis.
- Create a variety of plots using **ggplot2**, including histograms, scatter plots, and heatmaps.
- Apply these skills to analyze a realistic dataset (`biomed_data.csv`) containing information on 200 patients.

Why Learn R?

R is an essential tool for bioinformatics due to its flexibility, extensive libraries, and active community. It allows researchers to handle large datasets, perform complex statistical analyses, and create publication-quality visualizations.

2 The RStudio Interface

Pane	Purpose	Tip
Source/Editor (top-left)	Write & save scripts (<code>.R</code> , <code>.Rmd</code>)	Ctrl/Cmd + Shift + C toggles comments
Console (bottom-left)	Run commands interactively	↑/↓ scrolls through history
Environment / History (top-right)	View objects & past commands	Click  to remove objects
Files / Plots / Packages / Help	File nav, graphics, install pkgs, docs	Click a plot → <i>Export</i> to save

Tip: Use the shortcut **Ctrl + Enter** (Windows) or **Cmd + Enter** (Mac) to run a line of code from the Source Pane in the Console.

```
knitr::opts_knit$set(root.dir = "/workspaces/codespaces_RStudio/course_materials/R_basics")

# Set your working directory once per session
getwd()
```

```
[1] "/workspaces/codespaces_RStudio/practicals"
```

```
setwd("/workspaces/codespaces_RStudio/course_materials/R_basics")

# Install required libraries once:

install.packages(c("tidyverse", "viridis", "GGally"))
```

Installing packages into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

3 R Programming Basics

Writing Your First R Code

Let's start with some simple R commands. Open the R Console and try the following:

3.1 Console vs Editor

- Console for quick commands.
- Editor for scripts.
- Execute lines with Ctrl/Cmd + Enter.

```
# Basic arithmetic
2 + 2 # Addition
```

```
[1] 4
```

```
5 * 3 # Multiplication
```

```
[1] 15
```

```
10 / 2 # Division
```

```
[1] 5
```

Explanation Comments: Lines starting with # are comments and are ignored by R. Assignment: The <- operator assigns values to variables. You can also use = but <- is preferred in R. Execution: Type the code in the Console or write it in a script and run it. Exercise: Try assigning different values to x and y and observe how z changes.

3.2 Variable Assignment & Arithmetic

```
x <- 10    # arrow assignment  
y = 5      # equals also valid  
x
```

```
[1] 10
```

```
x + y      # addition
```

```
[1] 15
```

```
x * y      # multiplication
```

```
[1] 50
```

```
x ^ 2      # exponentiation
```

```
[1] 100
```

```
x > y      # logical test
```

```
[1] TRUE
```

Explanation Comments: Lines starting with # are comments and are ignored by R. Assignment: The <- operator assigns values to variables. You can also use = but <- is preferred in R for clarity and consistency. Arithmetic Operations: R supports basic arithmetic operations like addition (+), multiplication (*), and exponentiation (^). Logical Tests: Logical operators like > return TRUE or FALSE based on the comparison. Exercise: Assign different values to x and y and observe how the results of x + y and x * y change. Try other logical operators like <, ==, and != to compare x and y.

3.3 Data Types & Coercion

```
my_numeric <- 42.0
my_char     <- "Hello"
my_logical  <- TRUE

class(my_numeric) # "numeric"
```

```
[1] "numeric"
```

```
my_numeric <- as.character(my_numeric) # "42"
```

Explanation Data Types: R has several basic data types, including: numeric: Numbers with or without decimals. character: Text or strings. logical: Boolean values (TRUE or FALSE). Type Coercion: Functions like `as.character()` can convert one data type to another. For example, a numeric value can be converted to a character string. Exercise: Create variables of different data types (e.g., integer, factor) and use the `class()` function to check their types. Try coercing a character string to numeric using `as.numeric()` and observe the result.

3.4 Exploring Objects

```
v <- c(1,2,3,4,5)
str(v)           # structure
```

```
num [1:5] 1 2 3 4 5
```

```
summary(v)      # summary stats
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	2	3	3	4	5

```
length(v)       # 5
```

```
[1] 5
```

Explanation Vectors: The `c()` function combines values into a vector. Vectors are one of the most basic data structures in R. Functions: R provides built-in functions like `length()`, `sum()`, and `mean()` to operate on vectors. Exercise: Create a vector with at least 10 elements and calculate its sum, mean, and length. Try creating a character vector (e.g., `c("A", "B", "C")`) and observe what happens when you use `sum()` or `mean()` on it.

4 Exploring Objects & Structures

Vectors

```
v <- c(10, 20, 30, 40, 50)
```

```
v[1]      # Access the first element
```

```
[1] 10
```

```
v[2:4]    # Access elements from index 2 to 4
```

```
[1] 20 30 40
```

```
v[c(1, 5)] # Access the 1st and 5th elements
```

```
[1] 10 50
```

```
v[v > 25] # Subset elements greater than 25
```

```
[1] 30 40 50
```

Explanation Indexing: Use square brackets [] to access specific elements of a vector. Indexing in R starts at 1 (not 0). Subsetting: You can subset vectors using logical conditions (e.g., `v > 25`). Exercise: Create a vector of your choice and practice accessing individual elements and ranges. Use logical conditions to subset elements based on specific criteria (e.g., values less than 15).

Matrices

```
# Create a 3x3 matrix
```

```
m <- matrix(1:9, nrow = 3, byrow = TRUE)
```

```
m[2, 3]      # Access the element in the 2nd row, 3rd column
```

```
[1] 6
```

```
dim(m)           # Get the dimensions of the matrix (rows and columns)
```

```
[1] 3 3
```

```
m[1, ]           # Access the entire first row
```

```
[1] 1 2 3
```

```
m[, 2]           # Access the entire second column
```

```
[1] 2 5 8
```

Explanation Matrix Creation: The `matrix()` function creates a matrix. The `nrow` argument specifies the number of rows, and `byrow = TRUE` fills the matrix row-wise. Indexing: Use `[row, column]` to access specific elements. Leaving the row or column blank (e.g., `[1,]`) selects all elements in that dimension. Dimensions: The `dim()` function returns the dimensions of the matrix as a vector (number of rows and columns). Exercise: Create a 4x4 matrix with numbers from 1 to 16. Access the element in the 3rd row and 4th column. Extract the 2nd row and the 3rd column separately. Try adding two matrices of the same dimensions.

Lists

```
# Create a list containing a vector and a matrix
my_list <- list(a = v, b = m)
```

```
my_list$a         # Access the vector by name
```

```
[1] 10 20 30 40 50
```

```
my_list$b         # Access the matrix by name
```

```
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

```
my_list[[1]] # Access the first element of the list (vector)
```

```
[1] 10 20 30 40 50
```

```
my_list[[2]][2, 3] # Access the 2nd row, 3rd column of the matrix in the list
```

```
[1] 6
```

Explanation Lists: A list is a flexible data structure that can contain elements of different types (e.g., vectors, matrices, data frames). Accessing Elements: Use `$` to access elements by name or `[[]]` to access elements by position. You can also combine indexing to access specific parts of nested elements. Exercise: Create a list containing a numeric vector, a character vector, and a matrix. Access each element of the list using `$` and `[[]]`. Modify one of the elements in the list (e.g., change a value in the matrix). Add a new element to the list (e.g., a logical vector).

Data Frames

```
# Create a data frame with 3 columns 10 rows
df <- data.frame(gene = c("G1", "G2", "G3", "G4", "G5", "G6", "G7", "G8", "G9", "G10"),
                 exp = c(1.2, 3.4, 5.6, 7.8, 9.0, 2.3, 4.5, 6.7, 8.9, 10.1),
                 group = c("A", "B", "A", "B", "A", "B", "A", "B", "A", "B"),
                 stringsAsFactors = FALSE)
str(df)
```

```
'data.frame':  10 obs. of  3 variables:
 $ gene : chr  "G1" "G2" "G3" "G4" ...
 $ exp  : num  1.2 3.4 5.6 7.8 9 2.3 4.5 6.7 8.9 10.1
 $ group: chr  "A" "B" "A" "B" ...
```

```
summary(df)
```

gene	exp	group
Length:10	Min. : 1.200	Length:10
Class :character	1st Qu.: 3.675	Class :character
Mode :character	Median : 6.150	Mode :character
	Mean : 5.950	
	3rd Qu.: 8.625	
	Max. :10.100	


```
# Accessing data frame elements
# Accessing data frame elements
df$gene      # Access the "gene" column
```

```
[1] "G1" "G2" "G3" "G4" "G5" "G6" "G7" "G8" "G9" "G10"
```

```
df[1, ]      # Access the first row
```

```
  gene exp group
1   G1 1.2     A
```

```
df[ , 2]      # Access the second column
```

```
[1] 1.2 3.4 5.6 7.8 9.0 2.3 4.5 6.7 8.9 10.1
```

```
df[df$exp > 5, ] # Subset rows where exp is greater than 5
```

```
  gene exp group
3   G3 5.6     A
4   G4 7.8     B
5   G5 9.0     A
8   G8 6.7     B
9   G9 8.9     A
10  G10 10.1    B
```

```
df[df$group == "A", ] # Subset rows where group is "A"
```

```
  gene exp group
1   G1 1.2     A
3   G3 5.6     A
5   G5 9.0     A
7   G7 4.5     A
9   G9 8.9     A
```

```
df[df$exp > 5 & df$group == "A", ] # Subset rows where exp is greater than 5 and group is "A"
```

```

      gene exp group
3    G3  5.6     A
5    G5  9.0     A
9    G9  8.9     A

```

```

# Adding a new column
df$Pass <- ifelse(df$exp > 5, "Pass", "Fail")
df

```

```

      gene exp group Pass
1     G1  1.2     A Fail
2     G2  3.4     B Fail
3     G3  5.6     A Pass
4     G4  7.8     B Pass
5     G5  9.0     A Pass
6     G6  2.3     B Fail
7     G7  4.5     A Fail
8     G8  6.7     B Pass
9     G9  8.9     A Pass
10    G10 10.1     B Pass

```

Explanation Data Frames: A data frame is a table-like structure where each column can have a different data type (e.g., `numeric`, `character`). Accessing Columns: Use `$` to access a specific column by name. Subsetting Rows: Use logical conditions to filter rows based on specific criteria. Adding Columns: You can add new columns to a data frame using the `$` operator. In the example, a new column `Pass` is added based on the condition that `exp` is greater than 5. Exercise: Create a data frame with at least 5 columns and 10 rows. Access specific columns and rows using `$` and `[]`. Subset the data frame based on specific conditions (e.g., values greater than a certain threshold).

Factors

Factors are used to represent categorical data in R. They are important for statistical modeling and data analysis. They can be ordered or unordered and are useful for representing categorical variables in data frames.

```

# Create a factor variable
df$group <- factor(df$group, levels = c("A", "B"), labels = c("Group A", "Group B"))
df$group

```

```
[1] Group A Group B Group A Group B Group A Group B Group A Group B Group A
[10] Group B
Levels: Group A Group B
```

```
# Check the structure of the data frame
str(df)
```

```
'data.frame':  10 obs. of  4 variables:
 $ gene : chr  "G1" "G2" "G3" "G4" ...
 $ exp  : num  1.2 3.4 5.6 7.8 9 2.3 4.5 6.7 8.9 10.1
 $ group: Factor w/ 2 levels "Group A","Group B": 1 2 1 2 1 2 1 2 1 2
 $ Pass : chr  "Fail" "Fail" "Pass" "Pass" ...
```

```
# Convert a character vector to a factor
df$gene <- factor(df$gene)
df$gene
```

```
[1] G1  G2  G3  G4  G5  G6  G7  G8  G9  G10
Levels: G1 G10 G2 G3 G4 G5 G6 G7 G8 G9
```

```
# Check the levels of the factor
levels(df$gene)
```

```
[1] "G1"  "G10" "G2"  "G3"  "G4"  "G5"  "G6"  "G7"  "G8"  "G9"
```

```
# Convert a factor back to a character vector
df$gene <- as.character(df$gene)
df$gene
```

```
[1] "G1"  "G2"  "G3"  "G4"  "G5"  "G6"  "G7"  "G8"  "G9"  "G10"
```

Explanation Factors: Factors are used to represent categorical data in R. They are important for statistical modeling and data analysis. Creating Factors: Use the `factor()` function to create a factor variable. You can specify the levels and labels for better readability. Converting Factors: You can convert a factor back to a character vector using `as.character()`. Exercise: Create a factor variable for the group column in the data frame. Check the levels of the factor and convert it back to a character vector.

5 Wrangling with dplyr

Explanation of dplyr Verbs The dplyr package provides a set of functions (verbs) for data manipulation. These functions are intuitive and work seamlessly with data frames and tibbles.

```
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

Verb	Description	Example
<code>select()</code>	choose columns	<code>select(df, gene, expr)</code>
<code>filter()</code>	choose rows	<code>filter(df, expr > 6)</code>
<code>mutate()</code>	add / transform column	<code>mutate(df, log2expr = log2(expr))</code>
<code>arrange()</code>	sort rows	<code>arrange(df, desc(expr))</code>
<code>group_by()</code>	split into groups	<code>group_by(df, gene)</code>
<code>summarise()</code>	aggregate per group	<code>summarise(df, mean(expr))</code>

5.1 select(): Choose Columns

```
# Select specific columns
df_selected <- select(df, gene, expr)
df_selected
```

```
   gene  expr
1   G1  1.2
2   G2  3.4
3   G3  5.6
4   G4  7.8
5   G5  9.0
```

```

6      G6  2.3
7      G7  4.5
8      G8  6.7
9      G9  8.9
10     G10 10.1

```

Description: The `select()` function is used to choose specific columns from a data frame. Example: In the example above, only the gene and exp columns are selected from the data frame.

5.2 filter(): Choose Rows

```

# Filter rows based on a condition
df_filtered <- filter(df, exp > 5)
df_filtered

```

```

      gene  exp  group Pass
1      G3  5.6 Group A Pass
2      G4  7.8 Group B Pass
3      G5  9.0 Group A Pass
4      G8  6.7 Group B Pass
5      G9  8.9 Group A Pass
6     G10 10.1 Group B Pass

```

Description: The `filter()` function is used to subset rows based on logical conditions. Example: In the example above, only rows where exp is greater than 5 are selected. Exercise: Use `filter()` to extract rows where gene is “G1”. Try filtering rows based on multiple conditions (e.g., `exp > 5` and `group == “A”`).

5.3 mutate(): Add or Transform Columns

```

# Add a new column
df_mutated <- mutate(df, Pass = ifelse(exp > 5, "Pass", "Fail"))
df_mutated

```

```

      gene  exp  group Pass
1      G1  1.2 Group A Fail
2      G2  3.4 Group B Fail
3      G3  5.6 Group A Pass
4      G4  7.8 Group B Pass

```

5	G5	9.0	Group A	Pass
6	G6	2.3	Group B	Fail
7	G7	4.5	Group A	Fail
8	G8	6.7	Group B	Pass
9	G9	8.9	Group A	Pass
10	G10	10.1	Group B	Pass

Description: The `mutate()` function is used to add new columns or modify existing ones. Example: In the example above, a new column `Pass` is added based on the condition that `exp` is greater than 5. Exercise: Add a new column that calculates the square of the `exp` column. Try using `mutate()` to create a new column that categorizes `exp` into “Low” (≤ 5) and “High” (> 5).

5.4 `arrange()`: Sort Rows

```
# Sort rows by Score in descending order
df_arranged <- arrange(df, desc(exp))
df_arranged
```

	gene	exp	group	Pass
1	G10	10.1	Group B	Pass
2	G5	9.0	Group A	Pass
3	G9	8.9	Group A	Pass
4	G4	7.8	Group B	Pass
5	G8	6.7	Group B	Pass
6	G3	5.6	Group A	Pass
7	G7	4.5	Group A	Fail
8	G2	3.4	Group B	Fail
9	G6	2.3	Group B	Fail
10	G1	1.2	Group A	Fail

Description: The `arrange()` function is used to sort rows based on one or more columns. Example: In the example above, rows are sorted by `exp` in descending order. Exercise: Sort rows by `gene` in ascending order. Try sorting by multiple columns (e.g., first by `group` and then by `exp`).

5.5 `group_by()` and `summarise()`: Group and Aggregate

```
library(dplyr)

df %>%
  group_by(group) %>%
  summarise(mean_expr = mean(exp),
            count      = n())

# A tibble: 2 x 3
  group    mean_expr count
  <fct>      <dbl> <int>
1 Group A      5.84     5
2 Group B      6.06     5

# Group by Pass/Fail and calculate average expression
df_grouped <- df %>%
  mutate(Pass = ifelse(exp > 5, "Pass", "Fail")) %>%
  group_by(Pass) %>%
  summarise(mean_exp = mean(exp))
df_grouped

# A tibble: 2 x 2
  Pass    mean_exp
  <chr>      <dbl>
1 Fail      2.85
2 Pass      8.02
```

Description: The `group_by()` function is used to split the data into groups based on one or more columns. The `summarise()` function is then used to calculate summary statistics for each group. Example: In the example above, the data is grouped by gene, and the mean expression and count of rows are calculated for each group. Exercise: Group the data by group and calculate the mean expression for each group.

6 Reshaping with tidyr

The `tidyr` package is used for reshaping and tidying data. It provides functions like `pivot_longer()` and `pivot_wider()` to transform data between wide and long formats.

6.1 `pivot_longer()`: Wide to Long Format

```
# Example: Convert wide data to long format
library(tidyr)
wide <- data.frame(id = 1:2, A = c(5,7), B = c(2,3))
wide
```

```
  id A B
1  1 5 2
2  2 7 3
```

```
long <- wide %>%
  pivot_longer(cols = A:B,
               names_to = "marker",
               values_to = "value")
long
```

```
# A tibble: 4 x 3
   id marker value
<int> <chr> <dbl>
1     1 A     5
2     1 B     2
3     2 A     7
4     2 B     3
```

Description: The `pivot_longer()` function converts wide-format data into long-format data by gathering multiple columns into key-value pairs. Example: In the example above, columns A and B are gathered into a single column marker, with their values stored in the value column. Exercise: Create a wide data frame with 3 rows and 4 columns (e.g., id, X, Y, Z). Use `pivot_longer()` to convert it into long format. Rename the new columns to variable and measurement.

6.2 pivot_wider(): Long to Wide Format

```
# Example: Convert long data back to wide format
wide_again <- long %>%
  pivot_wider(names_from = marker,
              values_from = value)
wide_again
```



```
# A tibble: 2 x 3
  id     A     B
<int> <dbl> <dbl>
1     1     5     2
2     2     7     3
```

Description: The `pivot_wider()` function converts long-format data into wide-format data by spreading key-value pairs into multiple columns. Example: In the example above, the marker column is spread into separate columns A and B. Exercise: Take the long-format data from the previous exercise and convert it back to wide format. Try using `pivot_wider()` with a dataset that has multiple grouping variables.

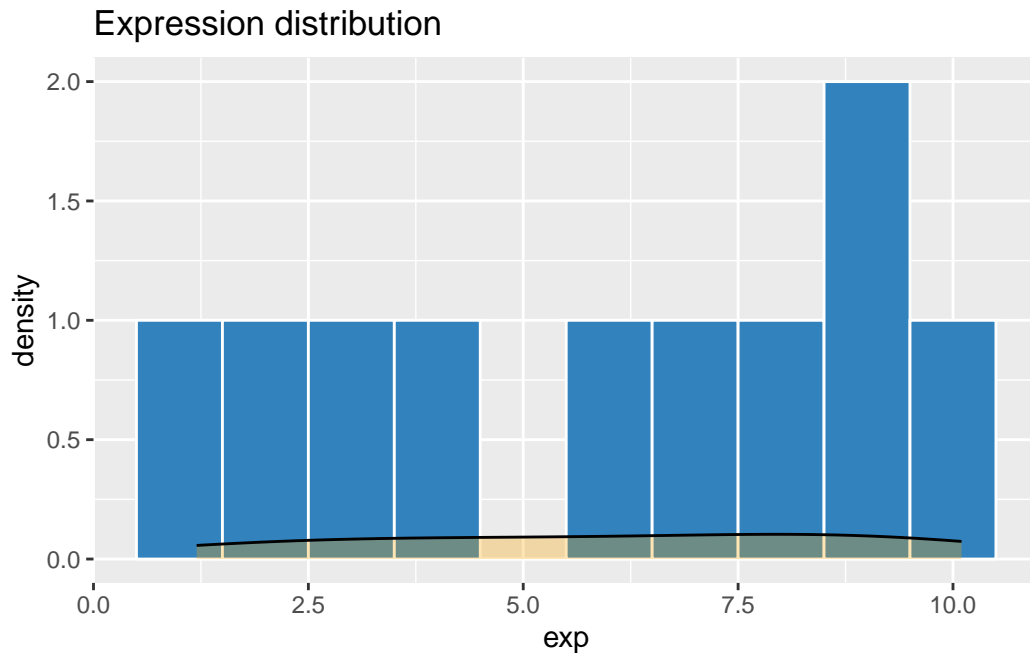
7 Visualising with ggplot2

The `ggplot2` package is a powerful tool for creating visualizations in R. It uses a layered grammar of graphics to build plots step by step.

```
library(ggplot2)
```

7.1 Histogram + Density Plot

```
ggplot(df, aes(exp)) +
  geom_histogram(binwidth = 1, fill = "#3182bd", colour = "white") +
  geom_density(alpha = .3, fill = "orange") +
  labs(title = "Expression distribution")
```

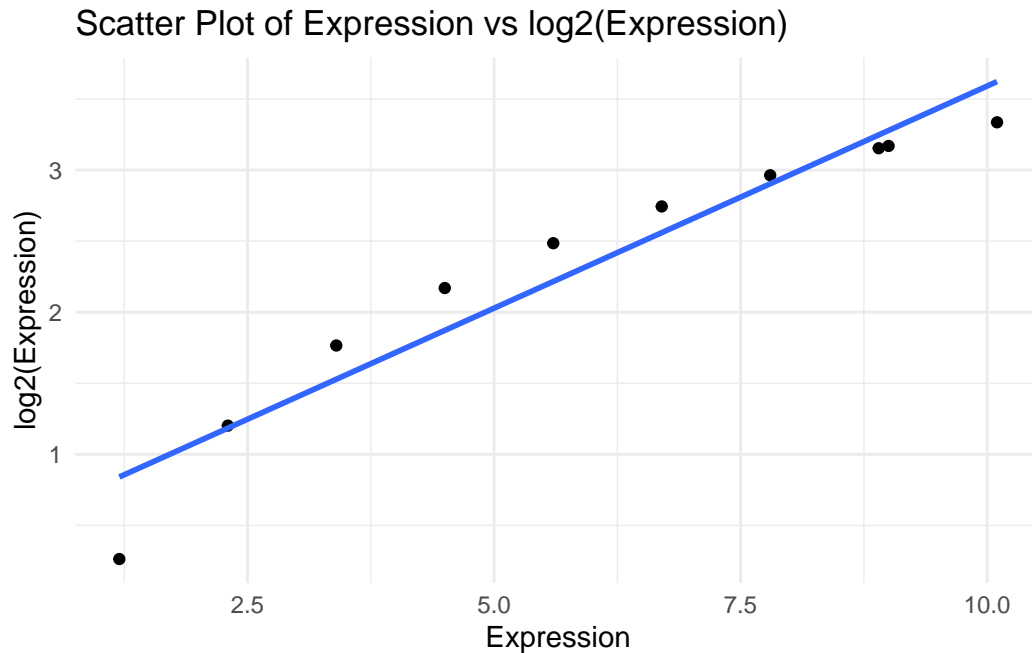


Description: - `geom_histogram()`: Creates a histogram to visualize the distribution of expression values. - `geom_density()`: Adds a density plot to show the distribution shape. - `binwidth`: Controls the width of the histogram bins. - `alpha`: Controls the transparency of the density plot. Exercise: - Create a histogram of the Age column with a binwidth of 5. - Add a density plot to the histogram. - Experiment with different fill colors and transparency levels.

7.2 Scatter Plot

```
# Create a scatter plot
ggplot(df, aes(exp, log2(exp))) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Scatter Plot of Expression vs log2(Expression)",
       x = "Expression",
       y = "log2(Expression)") +
  theme_minimal()
```

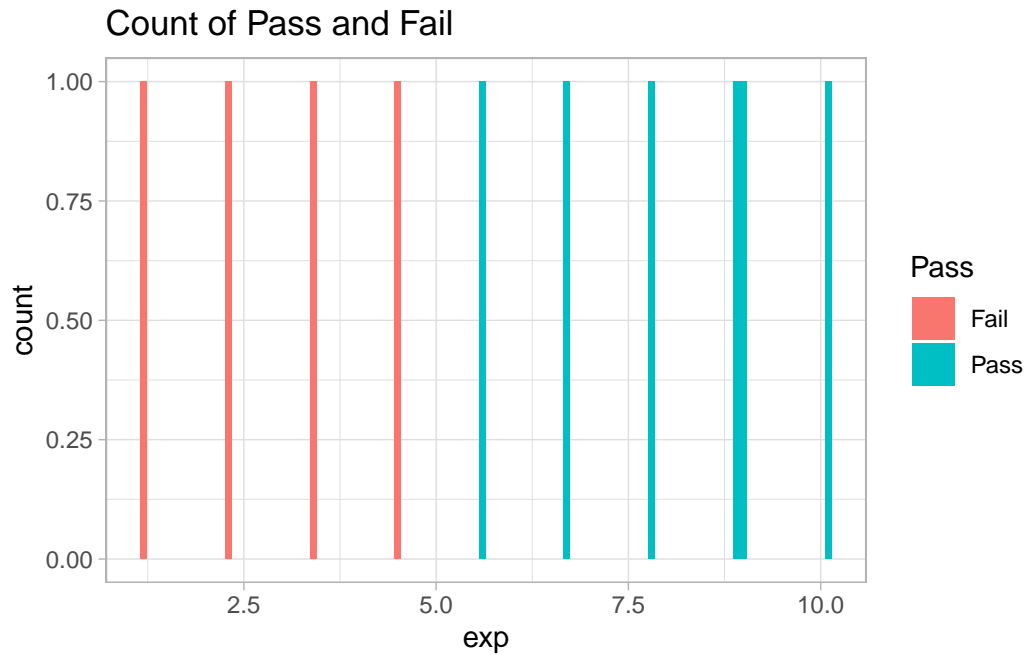
`geom_smooth()` using formula = 'y ~ x'



Description: `geom_point()`: Creates a scatter plot of expression vs log2(expression). `geom_smooth(method = "lm", se = FALSE)`: Adds a linear regression line without confidence intervals. `labs()`: Adds titles and labels to the axes. `theme_minimal()`: Applies a clean, minimal theme.

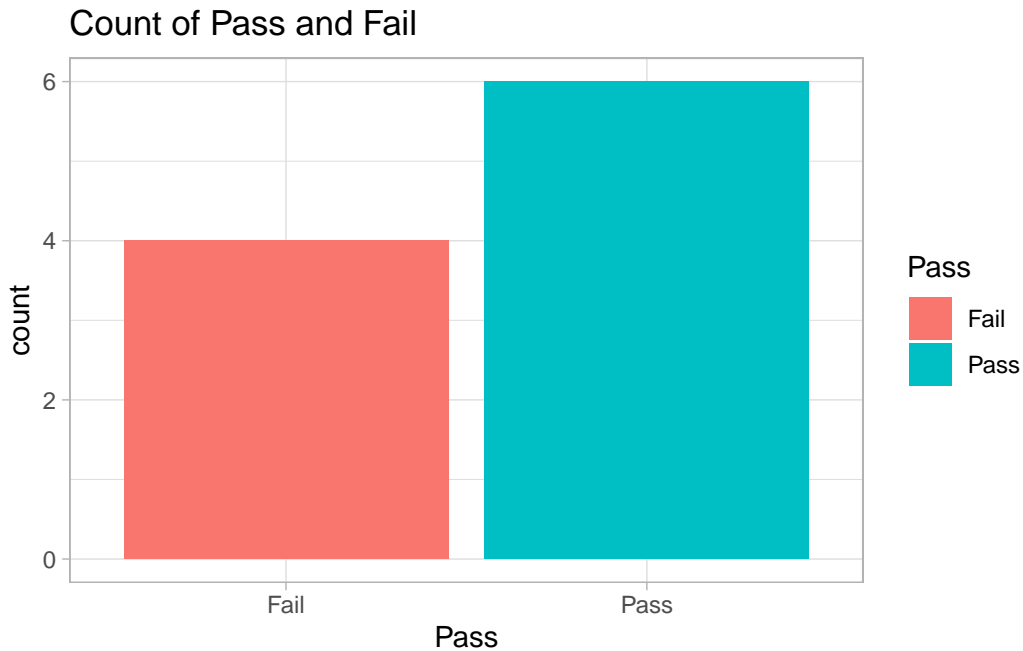
7.3 Bar Plot

```
# Create a bar plot
ggplot(df , aes(x = exp, fill = Pass)) +
  geom_bar() +
  theme_light() +
  labs(title = "Count of Pass and Fail")
```



Description: `geom_bar()`: Creates a bar plot. `aes(fill = Pass)`: Fills the bars with colors based on the Pass column. `theme_light()`: Applies a light theme. Exercise: Create a bar plot showing the count of Pass and Fail. Experiment with different themes (e.g., `theme_classic()`, `theme_dark()`).

```
ggplot(df, aes(x = Pass, fill = Pass)) +  
  geom_bar() +  
  theme_light() +  
  labs(title = "Count of Pass and Fail")
```



8 Full Workflow on biomed_data.csv

8.1 Load & Inspect

```
library(readr)
biomed <- read_csv("biomed_data.csv", show_col_types = FALSE)
glimpse(biomed)      # columns & types
```

```
Rows: 200
Columns: 14
$ patient_id <chr> "P1", "P2", "P3", "P4", "P5", "P6", "P7", "P8", "P9", "P10"~
$ sex        <chr> "Female", "Female", "Female", "Female", "Male", "Female", "~
$ age        <dbl> 64, 27, 32, 43, 28, 44, 43, 31, 56, 41, 49, 59, 26, 61, 71,~
$ group      <chr> "Treated", "Treated", "Treated", "Control", "Control", "Con~
$ region     <chr> "North", "North", "North", "South", "South", "North", "Sout~
$ dose       <chr> "Low", "High", "High", "Low", "Medium", "High", "High", "Me~
$ marker_A   <dbl> 5.64, 3.61, 3.87, 3.86, 4.31, 5.17, 5.59, 4.18, 2.14, 5.95,~
$ marker_B   <dbl> 14.07, 11.44, 11.98, 13.23, 13.29, 10.17, 14.07, 8.11, 7.78~
$ marker_C   <dbl> 49.52, 46.27, 51.05, 45.76, 53.96, 50.15, 53.02, 45.03, 48.~
$ marker_D   <dbl> 1.23, 1.81, 1.91, 1.12, 0.99, 1.24, 1.54, 1.31, 0.79, 1.04,~
$ marker_E   <dbl> 11.38, 11.62, 10.65, 10.73, 9.75, 8.69, 10.27, 10.29, 9.00,~
$ expression <dbl> 8.66, 12.54, 11.99, 11.27, 8.67, 10.80, 8.90, 8.97, 6.92, 1~
```

```
$ heart_rate <dbl> 69, 88, 84, 64, 91, 100, 64, 85, 92, 91, 99, 99, 94, 94, 63~
$ RBC_count <dbl> 4.47, 4.38, 3.99, 4.02, 5.32, 4.45, 4.73, 4.74, 4.95, 4.41,~
```

```
summary(biomed) # summary stats & NAs
```

```

patient_id      sex      age      group
Length:200      Length:200      Min.   :25.00      Length:200
Class :character Class :character 1st Qu.:38.00      Class :character
Mode  :character Mode  :character Median :49.00      Mode  :character
                        Mean   :50.09
                        3rd Qu.:61.25
                        Max.   :75.00

region          dose          marker_A      marker_B
Length:200      Length:200      Min.   :2.140      Min.   : 6.72
Class :character Class :character 1st Qu.:4.300      1st Qu.:11.10
Mode  :character Mode  :character Median :4.900      Median :12.58
                        Mean   :4.896      Mean   :12.43
                        3rd Qu.:5.525      3rd Qu.:13.89
                        Max.   :7.350      Max.   :18.09
                        NA's   :5

marker_C        marker_D        marker_E      expression
Min.   :35.35      Min.   :0.1400      Min.   : 5.850      Min.   : 4.000
1st Qu.:46.63      1st Qu.:0.8875      1st Qu.: 9.265      1st Qu.: 7.280
Median :49.93      Median :1.1200      Median :10.325      Median : 8.680
Mean   :49.89      Mean   :1.1838      Mean   :10.183      Mean   : 8.876
3rd Qu.:52.98      3rd Qu.:1.3550      3rd Qu.:11.110      3rd Qu.:10.340
Max.   :62.26      Max.   :2.9400      Max.   :15.040      Max.   :16.640
NA's   :5                                NA's   :5

heart_rate      RBC_count
Min.   : 57.00      Min.   :3.550
1st Qu.: 67.00      1st Qu.:4.265
Median : 77.00      Median :4.550
Mean   : 78.89      Mean   :4.554
3rd Qu.: 91.00      3rd Qu.:4.860
Max.   :103.00      Max.   :5.490
                        NA's   :5
```

Explanation - `read_csv()`: Reads a CSV file into a data frame. - `glimpse()`: Provides a quick overview of the data frame, including column names and types. - `summary()`: Provides summary statistics for each column, including mean, median, and number of missing values (NAs). - `show_col_types = FALSE`: Suppresses the display of column types in the output.

- Exercise: - Load the `biomed_data.csv` file and inspect its structure. - Check for missing values in the dataset. - Summarize the data to understand its distribution and key statistics. - Identify any potential outliers or anomalies in the data.

8.2 Key dplyr Demonstrations

a) Select & Filter

```
# Keep only patient_id, age, group, expression
biomed %>%
  select(patient_id, age, group, expression) %>%
  head(5)
```

```
# A tibble: 5 x 4
  patient_id age group expression
  <chr>      <dbl> <chr>      <dbl>
1 P1         64 Treated      8.66
2 P2         27 Treated     12.5
3 P3         32 Treated     12.0
4 P4         43 Control     11.3
5 P5         28 Control     8.67
```

```
# Keep only patient_id, age, group, expression
biomed_filtered <- biomed %>%
  select(patient_id, age, group, expression) %>%
  filter(age > 30 & group == "Treated")
biomed_filtered
```

```
# A tibble: 85 x 4
  patient_id age group expression
  <chr>      <dbl> <chr>      <dbl>
1 P1         64 Treated      8.66
2 P3         32 Treated     12.0
3 P8         31 Treated      8.97
4 P11        49 Treated      7.88
5 P15        71 Treated      9.45
6 P16        71 Treated      6.17
7 P17        54 Treated     12.7
8 P18        34 Treated     11.2
9 P20        73 Treated      7.29
10 P21        59 Treated      9.22
# i 75 more rows
```

Description: - `select()`: Keeps only the specified columns (`patient_id`, `age`, `group`, `expression`).
 - `filter()`: Filters rows where `age > 30` and `group` is "Treated". Exercise: Select only the `patient_id`, `group`, and `expression` columns. Filter rows where `expression` is greater than 50 and `group` is "Control".

```
# Patients older than 60 in Treated group
biomed %>%
  filter(age > 60, group == "Treated") %>%
  select(patient_id, age, group)
```

```
# A tibble: 29 x 3
  patient_id age group
  <chr>      <dbl> <chr>
1 P1         64 Treated
2 P15        71 Treated
3 P16        71 Treated
4 P20        73 Treated
5 P27        61 Treated
6 P30        71 Treated
7 P40        72 Treated
8 P61        66 Treated
9 P62        61 Treated
10 P64       66 Treated
# i 19 more rows
```

Description: - `filter()`: Filters rows where `age` is greater than 60 and `group` is "Treated".
 - `select()`: Keeps only the `patient_id`, `age`, and `group` columns. Exercise: - Filter patients with `expression > 10` and `group` "Control". - Select only the `patient_id`, `age`, and `expression` columns.

b) Mutate & Arrange

```
# Add a new column for age category and sort by expression
biomed_mutated <- biomed %>%
  mutate(age_category = ifelse(age > 50, "Senior", "Adult")) %>%
  arrange(desc(expression))
biomed_mutated
```

```
# A tibble: 200 x 15
  patient_id sex age group region dose marker_A marker_B marker_C marker_D
  <chr>      <chr> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>
```



```

1 P54      Fema~  59 Trea~ North High    3.31    12.8    NA      2.15
2 P127     Fema~  29 Trea~ East  High    5.19    15.1    51.3     2.05
3 P30      Fema~  71 Trea~ South High    4.57     9.95    43.8     1.28
4 P168     Fema~  70 Trea~ South Low     2.34    16.4    54.5     0.14
5 P91      Male   37 Trea~ North High    6.26    13.4    53.4     1.82
6 P140     Fema~  48 Trea~ North High    5.25    14.1    44.1     2.64
7 P10      Fema~  41 Cont~ North High    5.95    12.5    51.7     1.04
8 P165     Fema~  48 Trea~ North High    5.14    12.4    49.3     1.96
9 P79      Fema~  68 Trea~ East  High    4.62    12.5    53.6     1.99
10 P102     Male   62 Trea~ West  High    4.35    11.0    48.9     2.94
# i 190 more rows
# i 5 more variables: marker_E <dbl>, expression <dbl>, heart_rate <dbl>,
#   RBC_count <dbl>, age_category <chr>

```

Description: - `mutate()`: Adds a new column (`age_category`) that categorizes age into “Senior” (>50) and “Adult” (<=50). - `arrange()`: Sorts the data frame by expression in descending order.

Exercise: Create a new column that categorizes expression into “High” (>7) and “Low” (<=7). Sort the data by age in ascending order.

c) Group & Summarise

```

# Mean expression & count per group & dose
biomed %>%
  group_by(group, dose) %>%
  summarise(mean_expr = mean(expression, na.rm = TRUE),
            sd_expr    = sd(expression, na.rm = TRUE),
            n          = n())

```

``summarise()`` has regrouped the output.
i Summaries were computed grouped by group and dose.
i Output is grouped by group.
i Use ``summarise(.groups = "drop_last")`` to silence this message.
i Use ``summarise(.by = c(group, dose))`` for per-operation grouping (``?dplyr::dplyr_by``) instead.

```

# A tibble: 6 x 5
# Groups:   group [2]
  group dose mean_expr sd_expr n
  <chr> <chr>    <dbl>    <dbl> <int>
1 Control High      8.42      2.01    37

```

2 Control Low	8.63	1.79	34
3 Control Medium	7.89	1.88	34
4 Treated High	11.8	1.84	34
5 Treated Low	8.41	1.83	32
6 Treated Medium	7.90	1.77	29

Description: - `group_by()`: Groups the data by group and dose. - `summarise()`: Calculates the mean expression, standard deviation, and count of rows for each group and dose. Exercise: - Group the data by group and dose, and calculate the mean expression and standard deviation for each group.

```
# Group by group and calculate mean expression
biomed_summary <- biomed %>%
  group_by(group) %>%
  summarise(mean_expression = mean(expression, na.rm = TRUE),
            count = n())
biomed_summary
```

```
# A tibble: 2 x 3
  group mean_expression count
  <chr>          <dbl> <int>
1 Control         8.31   105
2 Treated         9.50    95
```

Description: `group_by()`: Groups the data by the group column. `summarise()`: Calculates the mean expression and the count of rows for each group. Exercise: Group the data by `age_category` and calculate the median expression for each category. Add a column to count the number of patients in each group.

```
# Frequency table of region
biomed %>%
  count(region) %>%
  arrange(desc(n))
```

```
# A tibble: 4 x 2
  region      n
  <chr> <int>
1 East    59
2 North   51
3 South   50
4 West    40
```

d) Full Pipeline Example

```
# Combine all steps into a single pipeline    <!-- filter(age > 30 & group == "Treatment") %>%
biomed_pipeline <- biomed %>%
  select(patient_id, age, group, expression) %>%

  mutate(age_category = ifelse(age > 50, "Senior", "Young")) %>%
  group_by(age_category) %>%
  summarise(mean_expression = mean(expression, na.rm = TRUE),
            count = n()) %>%
  arrange(desc(mean_expression))
biomed_pipeline
```

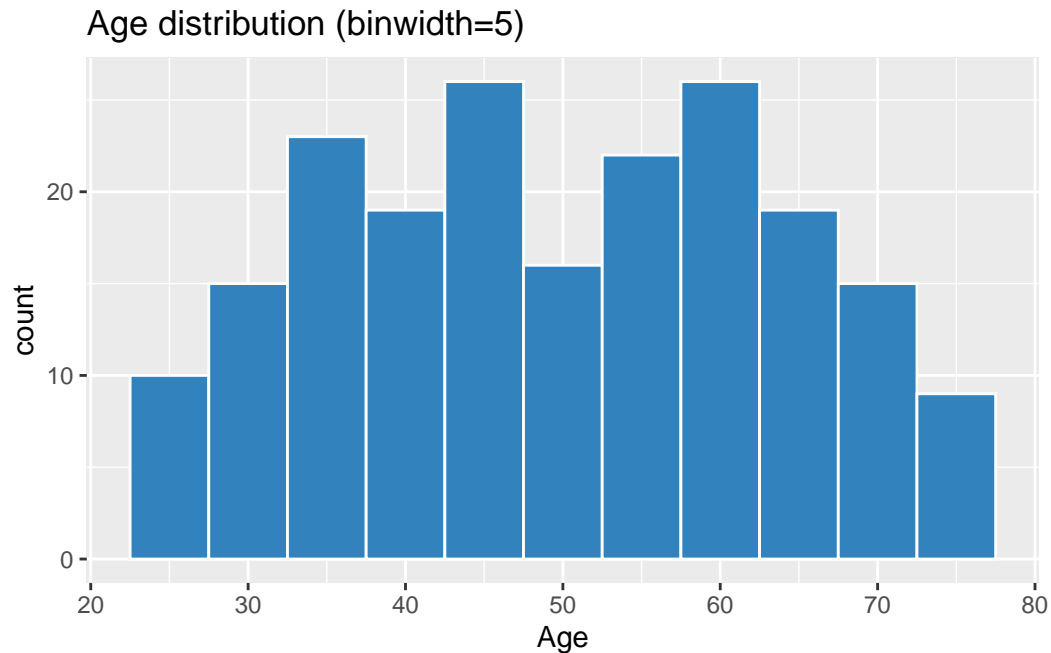
```
# A tibble: 2 x 3
  age_category mean_expression count
  <chr>         <dbl>    <int>
1 Young              9.37     101
2 Senior             8.36      99
```

Description: This pipeline combines `select()`, `filter()`, `mutate()`, `group_by()`, `summarise()`, and `arrange()` into a single workflow. Exercise: Modify the pipeline to include only patients with expression > 50. Add a step to calculate the standard deviation of expression for each age_category.

8.3 Extensive ggplot2 Gallery

Histogram of Age

```
ggplot(biomed, aes(age)) +
  geom_histogram(binwidth = 5, fill = "#3182bd", colour = "white") +
  labs(title = "Age distribution (binwidth=5)", x = "Age")
```

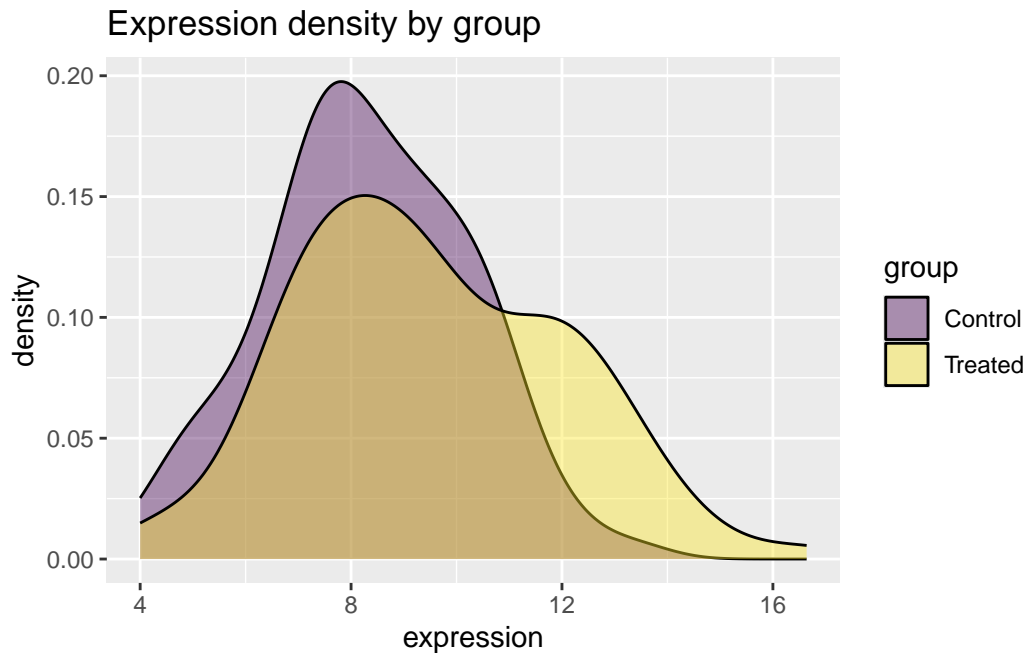


Explanation - `geom_histogram()`: Creates a histogram to visualize the distribution of age values. - `binwidth`: Controls the width of the histogram bins. - `labs()`: Adds titles and labels to the axes. - Exercise: - Create a histogram of expression with a binwidth of 10. - Add a density plot to the histogram. - Experiment with different fill colors and transparency levels.

Density of Expression by Group

```
ggplot(biomed, aes(expression, fill = group)) +
  geom_density(alpha = .4) +
  scale_fill_viridis_d() +
  labs(title = "Expression density by group")
```

Warning: Removed 5 rows containing non-finite outside the scale range (``stat_density()``).



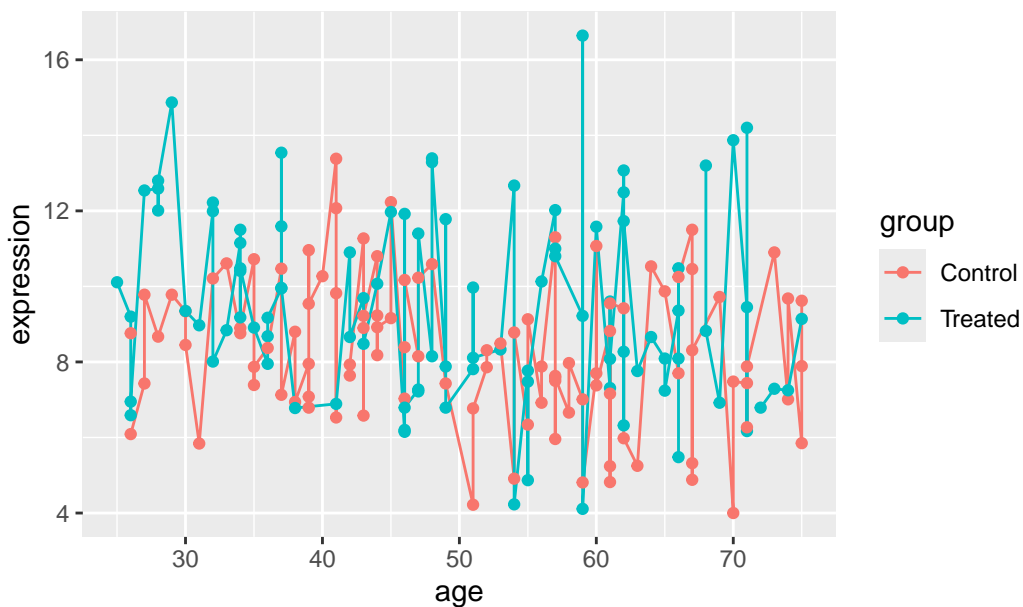
Explanation - `geom_density()`: Creates a density plot to visualize the distribution of expression values by group. - `alpha`: Controls the transparency of the density plot. - `scale_fill_viridis_d()`: Applies a color palette for better visibility. - Exercise: - Create a density plot of `heart_rate` by region. - Experiment with different fill colors and transparency levels.

a) Line Plot

```
# Line plot showing trends over time
ggplot(biomed, aes(x = age, y = expression, group = group, colour = group)) +
  geom_line() +
  geom_point() +
  labs(title = "Expression over Age by Group")
```

Warning: Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).

Expression over Age by Group

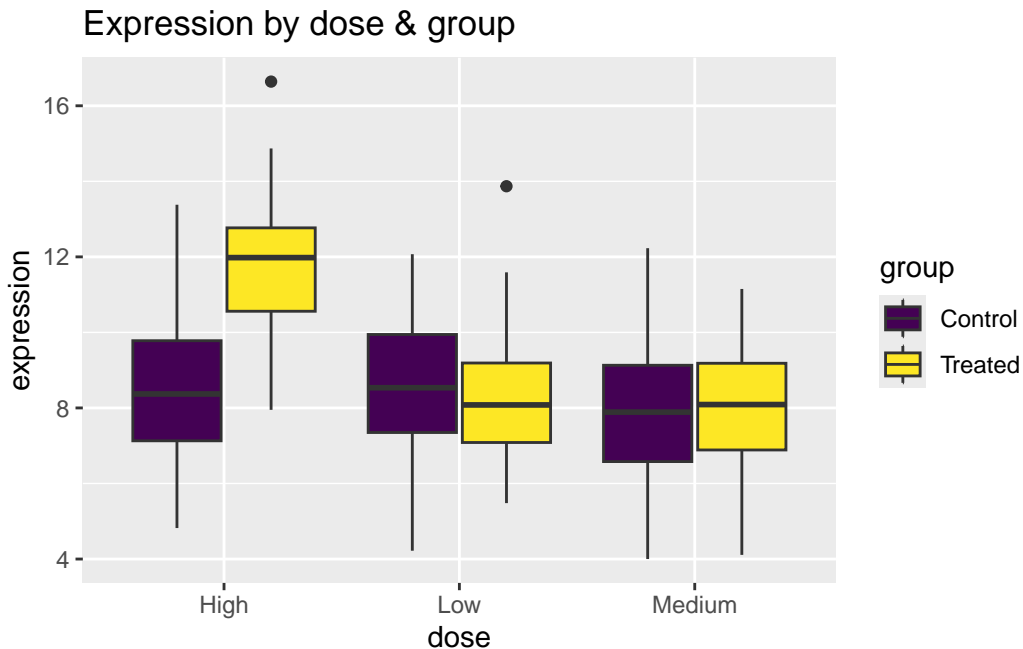


xplanation - `geom_line()`: Creates a line plot to visualize trends over time (or age). - `geom_point()`: Adds points to the line plot for better visibility. - `group`: Groups the data by the group variable. - `colour`: Colors the lines and points based on the group variable. - Exercise: - Create a line plot of `heart_rate` over age by region.

b) Boxplot: Expression by Dose & Group

```
ggplot(biomed, aes(dose, expression, fill = group)) +
  geom_boxplot(position = position_dodge(width = .8)) +
  scale_fill_viridis_d() +
  labs(title = "Expression by dose & group")
```

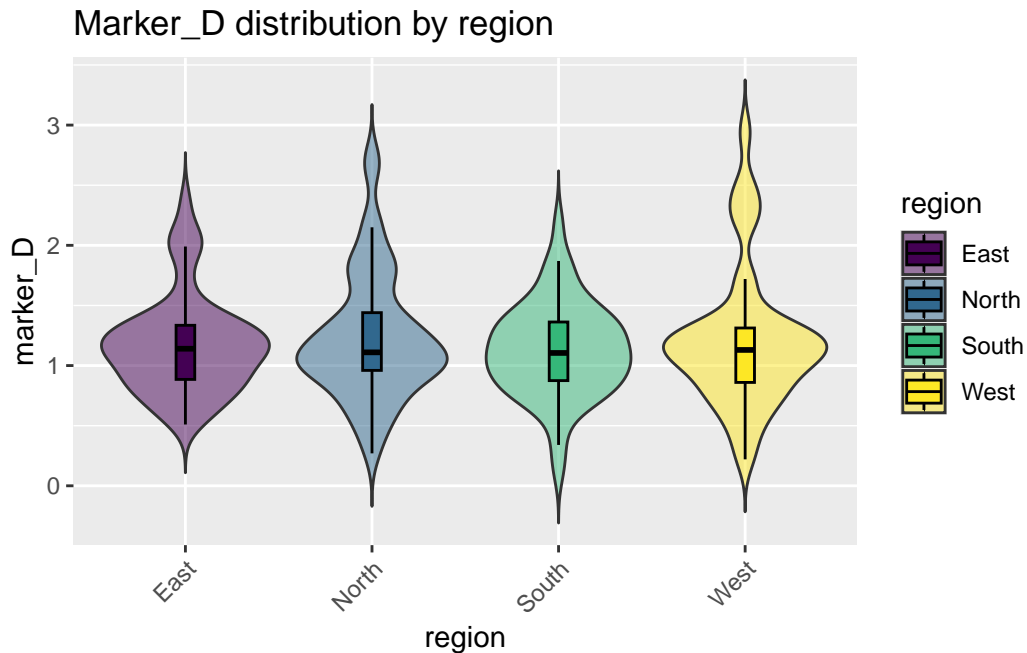
Warning: Removed 5 rows containing non-finite outside the scale range (``stat_boxplot()``).



Explanation - `geom_boxplot()`: Creates a boxplot to visualize the distribution of expression values by dose and group. - `position_dodge()`: Adjusts the position of the boxplots to avoid overlap. - `scale_fill_viridis_d()`: Applies a color palette for better visibility. Exercise: - Create a boxplot for marker_A by region. - Add jittered points to the boxplot to show individual data points.

c) Violin: Marker_D by Region

```
ggplot(biomed, aes(region, marker_D, fill = region)) +
  geom_violin(trim = FALSE, alpha = .5) +
  geom_boxplot(width = .1, colour = "black", outlier.shape = NA) +
  scale_fill_viridis_d() +
  labs(title = "Marker_D distribution by region") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Explanation - `geom_violin()`: Creates a violin plot to visualize the distribution of `marker_D` values by region. - `geom_boxplot()`: Adds a boxplot inside the violin plot for better summary statistics. - `trim = FALSE`: Ensures the violin plot is not trimmed at the tails. - `theme(axis.text.x = element_text(angle = 45, hjust = 1))`: Rotates x-axis labels for better readability. Exercise: - Create a violin plot for `marker_E` by group. - Add a boxplot inside the violin plot to show summary statistics.

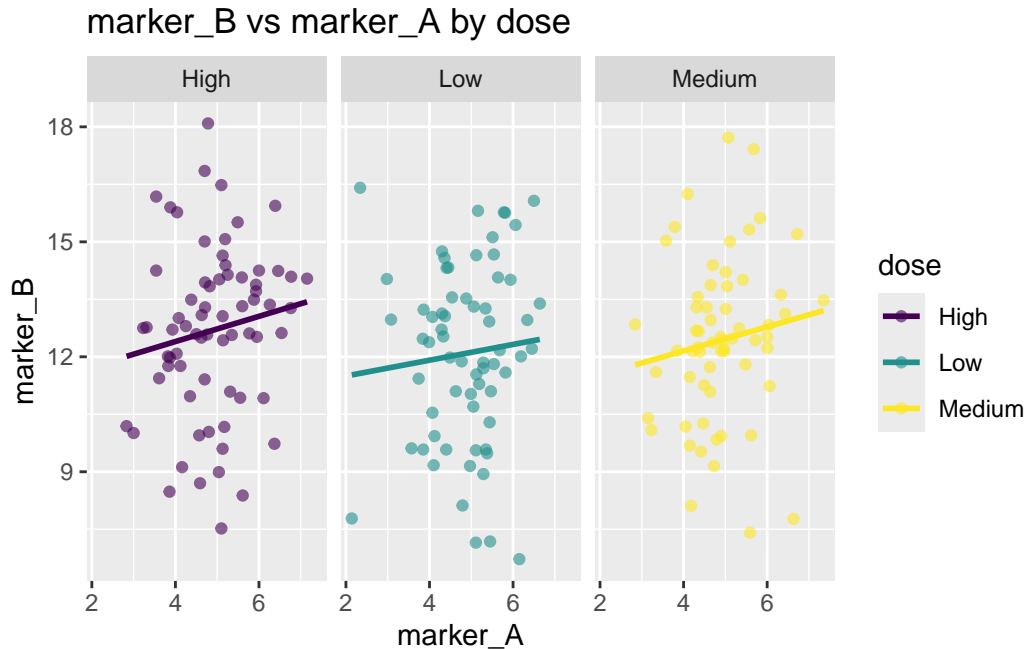
d) Scatter: `marker_B` vs `marker_A`, Faceted by Dose

```
ggplot(biomed, aes(marker_A, marker_B, colour = dose)) +
  geom_point(alpha = .6) +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ dose) +
  scale_colour_viridis_d() +
  labs(title = "marker_B vs marker_A by dose")
```

``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 5 rows containing non-finite outside the scale range (``stat_smooth()``).

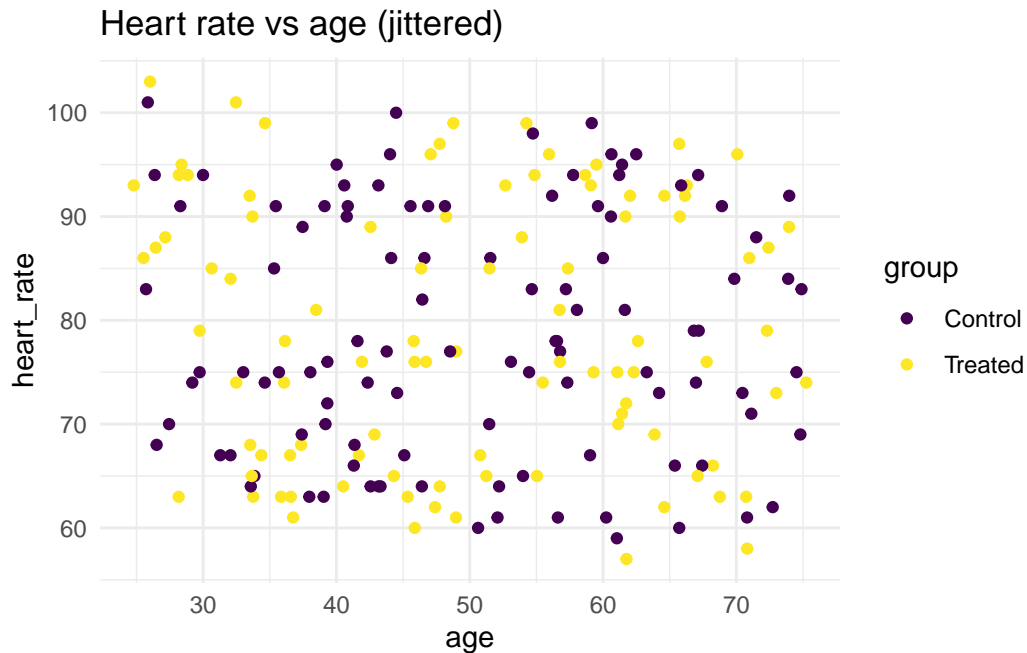
Warning: Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).



Explanation - `geom_point()`: Creates a scatter plot of marker_B vs marker_A. - `geom_smooth(method = "lm", se = FALSE)`: Adds a linear regression line without confidence intervals. - `facet_wrap(~ dose)`: Creates separate panels for each dose level. - `scale_colour_viridis_d()`: Applies a color palette for better visibility. Exercise: - Create a scatter plot of heart_rate vs age, colored by group. - Add a linear regression line to the scatter plot.

e) Jittered Scatter: Heart Rate vs Age

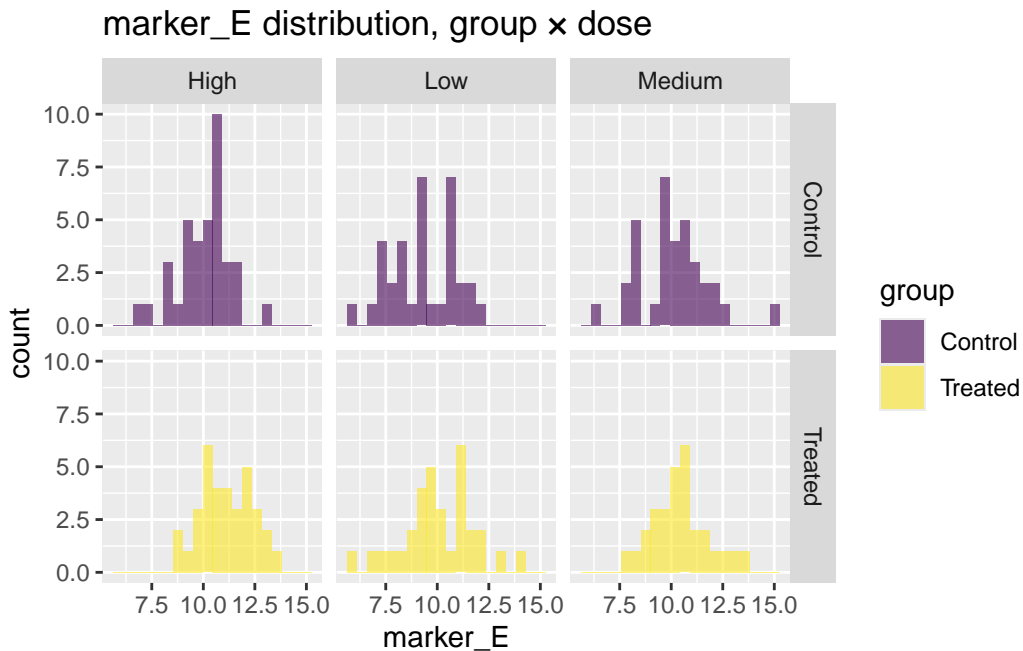
```
ggplot(biomed, aes(age, heart_rate, colour = group)) +
  geom_jitter(width = .5, height = 0) +
  scale_colour_viridis_d() +
  labs(title = "Heart rate vs age (jittered)") +
  theme_minimal()
```



Explanation - `geom_jitter()`: Creates a scatter plot with jitter to avoid overplotting. - `width` and `height`: Control the amount of jitter in the x and y directions. - `theme_minimal()`: Applies a clean, minimal theme. Exercise: - Create a jittered scatter plot of `marker_C` vs `age`, colored by `region`.

f) Facet Grid: `marker_E` by `Group` & `Dose`

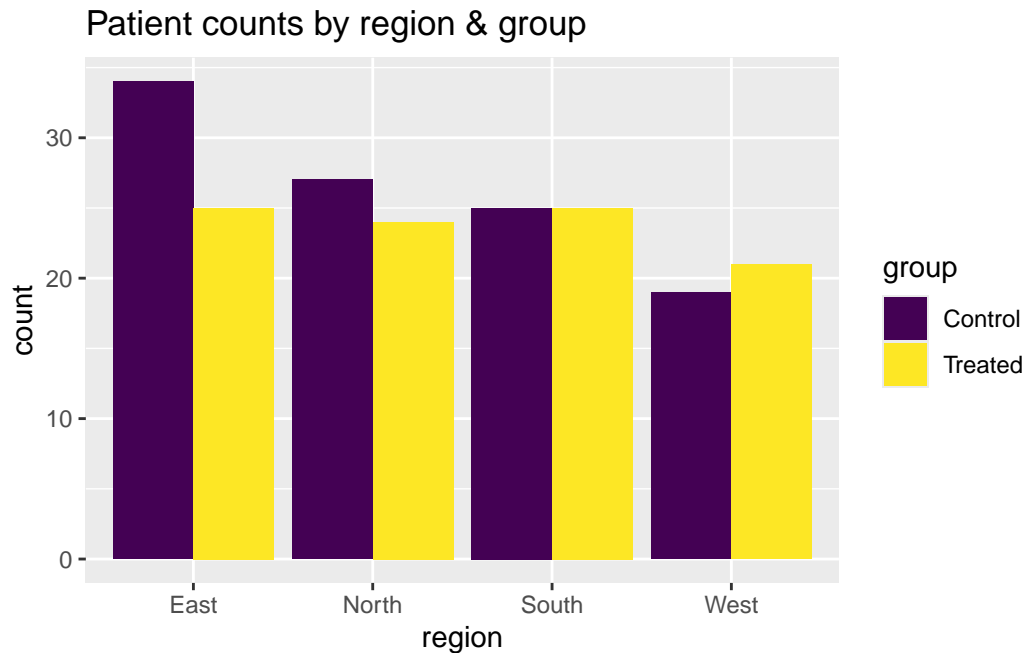
```
ggplot(biomed, aes(marker_E, fill = group)) +
  geom_histogram(bins = 20, alpha = .6) +
  facet_grid(group ~ dose) +
  scale_fill_viridis_d() +
  labs(title = "marker_E distribution, group × dose")
```



Explanation - `facet_grid(group ~ dose)`: Creates a grid of plots based on group and dose. - `geom_histogram(bins = 20, alpha = .6)`: Creates histograms for marker_E with 20 bins. - `scale_fill_viridis_d()`: Applies a color palette for better visibility. Exercise: - Create a facet grid of marker_A by region and group.

g) Bar Chart: Count per Region & Group

```
ggplot(biomed, aes(region, fill = group)) +
  geom_bar(position = "dodge") +
  scale_fill_viridis_d() +
  labs(title = "Patient counts by region & group")
```



Explanation - `geom_bar(position = "dodge")`: Creates a bar chart showing counts of patients by region and group. - `scale_fill_viridis_d()`: Applies a color palette for better visibility. - `labs(title = "Patient counts by region & group")`: Adds a title to the plot. Exercise: - Create a bar chart showing counts of patients by age category and group. h) Pairwise Scatterplot Matrix

```
library(GGally)
# Create a pairwise scatterplot matrix
ggpairs(biomed, columns = c("marker_A", "marker_B", "marker_C", "marker_D"),
        aes(colour = group, alpha = 0.5)) +
  theme_minimal() +
  labs(title = "Pairwise scatterplot matrix")
```

Warning: Removed 5 rows containing non-finite outside the scale range (``stat_density()``).

Warning: Removed 5 rows containing missing values

Warning: Removed 10 rows containing missing values

Warning: Removed 5 rows containing missing values

Warning: Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).

Warning: Removed 5 rows containing missing values

Warning: Removed 10 rows containing missing values or values outside the scale range (``geom_point()``).

Warning: Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).

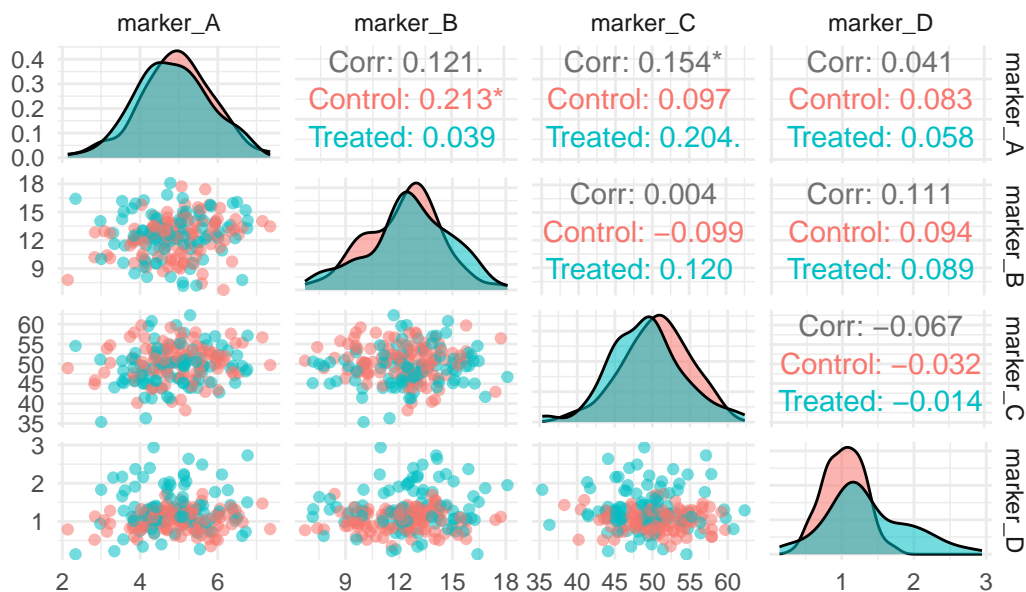
Warning: Removed 5 rows containing non-finite outside the scale range (``stat_density()``).

Warning: Removed 5 rows containing missing values

Warning: Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).

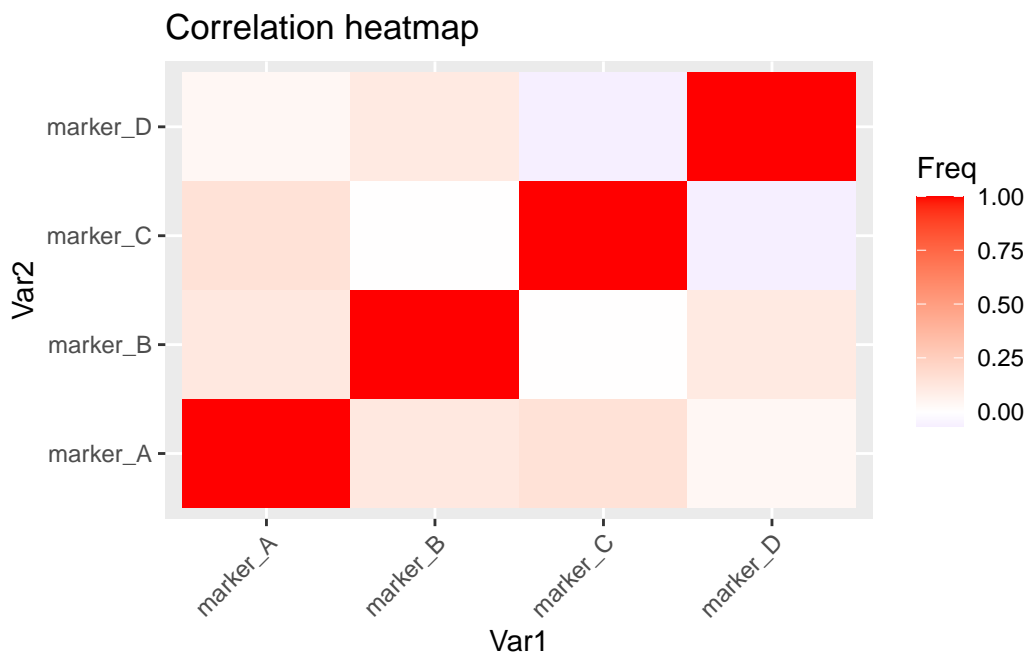
Removed 5 rows containing missing values or values outside the scale range (``geom_point()``).

Pairwise scatterplot matrix



Explanation - `ggpairs()`: Creates a matrix of scatterplots for selected columns. - `aes(colour = group, alpha = 0.5)`: Colors the points based on the group and sets transparency. - `theme_minimal()`: Applies a clean, minimal theme. - `labs(title = "Pairwise scatterplot matrix")`: Adds a title to the plot. Exercise: - Create a pairwise scatterplot matrix for heart_rate, RBC_count, and expression. i) Correlation Matrix

```
# Compute correlation matrix
cor_matrix <- cor(biomed[, c("marker_A", "marker_B", "marker_C", "marker_D")], use = "pairwi
# Convert to long form
library(tidyr)
# Convert correlation matrix to long format without using rownames_to_column
Long <- as.data.frame(as.table(cor_matrix))
# Plot heatmap
ggplot(Long, aes(Var1, Var2, fill = Freq)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
                      midpoint = 0) +
  labs(title = "Correlation heatmap") +
  theme(axis.text.x = element_text(angle=45, hjust=1))
```



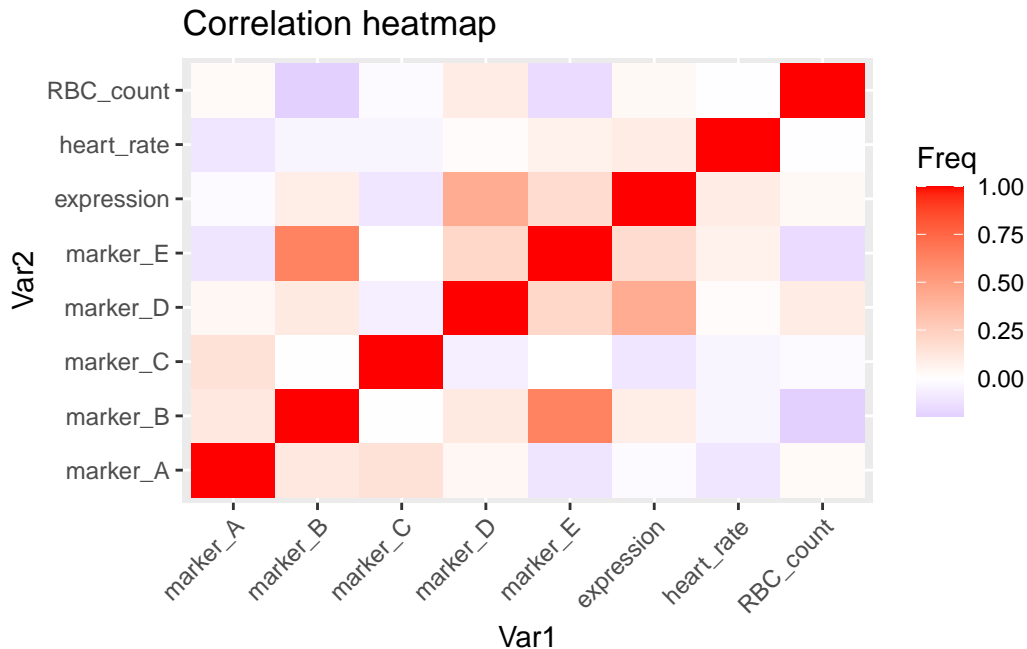
Explanation - `cor()`: Computes the correlation matrix for selected columns. - `as.table()`: Converts the correlation matrix to a table format. - `ggplot()`: Creates a heatmap of the correlation matrix. - `geom_tile()`: Creates a heatmap of the correlation

matrix. - `scale_fill_gradient2()`: Sets the color gradient for the heatmap. - `labs(title = "Correlation heatmap")`: Adds a title to the plot. - `theme(axis.text.x = element_text(angle=45, hjust=1))`: Rotates x-axis labels for better readability. Exercise: - Create a correlation matrix for `heart_rate`, `RBC_count`, and `expression`. - Plot the correlation matrix as a heatmap.

j) Correlation Heatmap of Numeric Variables

```
# Compute correlation matrix
nums <- biomed %>%
  select(marker_A, marker_B, marker_C, marker_D,
          marker_E, expression, heart_rate, RBC_count) %>%
  cor(use = "pairwise.complete.obs")

# Convert to long form
library(tidyr)
# corr_long <- as.data.frame(nums) %>%
#   rownames_to_column("var1") %>%
#   pivot_longer(-var1, names_to="var2", values_to="corr")
#
# # Plot heatmap
# ggplot(corr_long, aes(var1, var2, fill = corr)) +
#   geom_tile() +
#   scale_fill_gradient2(low = "blue", mid = "white", high = "red",
#                         midpoint = 0) +
#   labs(title = "Correlation heatmap") +
#   theme(axis.text.x = element_text(angle=45, hjust=1))
library(tidyr)
Long <- as.data.frame(as.table(nums))
# Plot heatmap
ggplot(Long, aes(Var1, Var2, fill = Freq)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red",
                      midpoint = 0) +
  labs(title = "Correlation heatmap") +
  theme(axis.text.x = element_text(angle=45, hjust=1))
```



Explanation - `cor()`: Computes the correlation matrix for selected columns. - `rownames_to_column()`: Converts row names to a column for easier plotting. - `pivot_longer()`: Reshapes the correlation matrix into long format. - `geom_tile()`: Creates a heatmap of the correlation matrix. - `scale_fill_gradient2()`: Sets the color gradient for the heatmap. - `labs(title = "Correlation heatmap")`: Adds a title to the plot. Exercise: - Create a correlation matrix for `heart_rate`, `RBC_count`, and `expression`. - Plot the correlation matrix as a heatmap.

References & Resources

Below is a curated list of books, websites, cheat-sheets and tutorials to deepen your understanding of R, the tidyverse, and biomedical data analysis.

Books

- **R for Data Science**
Wickham, H. & Grolemund, G. (2016). *R for Data Science*. O'Reilly Media.
Online: <https://r4ds.had.co.nz/> :contentReferenceoaicite:0

- **ggplot2: Elegant Graphics for Data Analysis**
Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer.
Online reference: <https://ggplot2.tidyverse.org/> :contentReferenceoaicite:1
 - **Advanced R**
Wickham, H. (2019). *Advanced R*. Chapman & Hall/CRC.
Online: <https://adv-r.hadley.nz/> :contentReferenceoaicite:2
-

Official Documentation

- **tidyverse** (“meta-package” including dplyr, tidyr, ggplot2, readr, etc.)
<https://www.tidyverse.org/> :contentReferenceoaicite:3
 - **dplyr reference**
<https://dplyr.tidyverse.org/reference/> :contentReferenceoaicite:4
 - **tidyr reference**
<https://tidyr.tidyverse.org/reference/> :contentReferenceoaicite:5
 - **ggplot2 reference**
<https://ggplot2.tidyverse.org/reference/> :contentReferenceoaicite:6
 - **readr reference**
<https://readr.tidyverse.org/reference/> :contentReferenceoaicite:7
-

Cheat-Sheets & Quick Guides

- **RStudio Cheat-Sheets**
Download PDF versions for dplyr, tidyr, ggplot2, and more:
<https://www.rstudio.com/resources/cheatsheets/> :contentReferenceoaicite:8
 - **Tidyverse Style Guide**
Conventions for writing tidyverse-style R code:
<https://style.tidyverse.org/> :contentReferenceoaicite:9
-

Online Tutorials & Courses

- **Swirl:** Interactive R lessons in the R console
<https://swirlstats.com/> :contentReferenceoaicite:10
 - **Coursera: R Programming** (Johns Hopkins University)
<https://www.coursera.org/learn/r-programming> :contentReferenceoaicite:11
 - **Datacamp: Introduction to R**
<https://www.datacamp.com/courses/free-introduction-to-r> :contentReferenceoaicite:12
-

Biomedical - Specific Resources

- **Bioconductor:** R packages for bioinformatics
<https://www.bioconductor.org/> :contentReferenceoaicite:13
 - **Bioconductor Workflow Guides**
E.g. RNA-seq, single-cell analyses:
https://www.bioconductor.org/packages/release/BiocViews.html#___Workflow :contentReferenceoaicite:14
-

Community & Help

- **RStudio Community**
<https://community.rstudio.com/> :contentReferenceoaicite:15
- **Stack Overflow (R tag)**
<https://stackoverflow.com/questions/tagged/r> :contentReferenceoaicite:16
- **R-Bloggers** (aggregated R tutorials and news)
<https://www.r-bloggers.com/> :contentReferenceoaicite:17